S-Base 1.00 - Release note

There have been some changes to S-Base since the manuals went to press. Most changes will only affect more advanced users.

This release note details the differences between the version of S-Base described in the manuals and version 1.0.

The contents of this release note are as follows:

1. Changes
2. Additions
3. Deletions
4. Comments
5. Configuration

1. Changes

i) Raw events

Raw events which do not related to a card or table (e.g. a screen mode change message) are normally sent to the default handler set. S-Base v1.0allows these events to be sent to a handler set called 'raw'. If this handler set does not have a handler for the particular raw event type, or the handler returns FALSE (indicating the event was not handled), then event is then passed onto the default handler set as explained in the Tutorial.

Because of the performance penalty of supporting raw events not related to a window, this feature is enabled by the configuration option 'Support non-window raw events'. Disabling this feature will cause all non-window raw events to be processed by S-Base and not first passed to either the 'raw' or default handler set.

Note that raw handler must return a value indicating whether the event was processed or not (TRUE if processed, else FALSE). You must return FALSE if the event was not processed, since it would otherwise be ignored. You must not return TRUE if the event is one which S-Base should see: for example, a mode change message must not be claimed in this way. Neither should a null event be claimed, as this will stop S-Base from operating it's background tasks.

2. Additions

i) menu clear

This command removes the current wimp menu from the screen, whatever it maybe. It is useful in applications when a keyboard short-cut is activated: in this situation any menu on the screen is generally removed.

ii) card element show <handle>, {<element> }

S-Base 1.0 supports hiding of card elements. Elements which are to be controlled in this way should be in the 'entry' area. Such elements will not appear when the card is displayed on the screen. In order to make them appear, this command is provided.

Card elements can be grouped into 'entries' as with reports. For example, if entry area element called 'invoice' appears before a group of prompt elements, then all the prompts can be shown by showing just the entry area.

An example of the use of this command is given in !Video, in the issue/reserve section (format 'issue reserve card'). There are three groups of prompts ('entry set 1...3') overlaid on each other. The procedures 'setup_read only' and 'setup_read write' in the 'circsys' program show and hide the prompts as required.

Note that fields and selectors should not be hidden (this is not supported inS-Base 1.0).

iii) card element hide <handle>, {<element> }

This command is the opposite of 'card element show'. It hides card elements or groups of elements.

iv) save to <fname>, <type>, <estsize>, <saver>
[, [<sender>] [, [<printer>] [, <result>]]]

This command is provided to allow saving of data via the normal RISC OS drag and drop method. This command pops up the standard 'save as' dialogue containing an icon representing the file to be saved. The user can drag this icon to the destination application or filer window.

<fname> is the suggested filename to appear in the dialogue

<type> is the file type, for example:

&fff : text
&ff9 : sprite
&aff : drawfile

This determines the icon which will appear in the dialogue.

<estsize> is the estimated size of the file to be saved. Where this is not known, 0 can be used. Supplying a reasonably accurate number here can increase performance when saving to some applications via ram transfer (is not via a filing system).

<saver> is a procedure to handle saving of the data to a file.

<sender> is a procedure to handle sending of the data via ram transfer. If this is not specified, saving will be done via a temporary file using<saver>.

<printer> is a procedure to handle printing of the data. If this is not specified, printing will be done via a temporary file using <saver>.

If the user decided to save the data, then one of the three 'handler 'procedures will be called. Only the 'saver' procedure must be specified: the other two are optional. However, it is common practice to support ram transfer so you should provide the 'sender' procedure where possible, even if a large memory overhead is required.

The form of the three handler procedures is shown in the example below,which saves a text file consisting of "S-Base v1.0". Note that when saving to a file your application must set the file type, even though it was specified in the 'save to' command.

```
def proc saver (fname)
  {the data to a file called fname. Then returns TRUE (non-zero) if all ok, FALSE (0) if not }

  | open the file
  local fd = @openout fname

  | write the text to the file
  bput fd, "S-Base v1.0"
```

```
  | close the file
  close fd

  | set the file type
  oscli "settype " + fname + " text"

  | all ok so return TRUE
  return TRUE
enddef

def proc sender
  {the data via ram transfer. Then returns TRUE (non-zero) if all ok, FALSE (0) if not }

  local b

  | put the text in a buffer
  buffer new b
  bput b, "S-Base v1.0"

  {waste space at the end of the buffer. We must do this since writing into buffers will cause the
buffer to expand in 4k steps, for speed/efficiency }
  buffer set ext b, @ptr b

  {the buffer. You can do this with any number of 'buffer send 'commands each sending any
amount of data. S-Base handles the complexities of sending the data to the destination
application. 'buffer send' is described below }
  buffer send b

  | dispose of the buffer
  buffer dispose b

  | all ok so return TRUE
  return TRUE
enddef

def proc printer (fname)
  {- print the file directly, or
    - save the file to fname
```

```
then:
  return -1: if the file was printed
        -2: if an error encountered while printing/saving
        or: if saved to a file, the file type of the file saved

  Note that printing the file via the RISC OS printer drivers is described in the RISC OS
Programmers' Reference Manual, and is beyond the scope of this release note }

  | use the 'saver' procedure to save the file
  local ok = proc saver (fname)

  | if something went wrong, return the code for 'error'
  if !ok then return -2

  | otherwise return the type of file saved, here text
  return &fff

enddef

| start the save operation
save to "test file", 0xfff, 10, saver, sender, printer
```

You can copy the above code into an application and try it out if you wish.The 'printer' procedure shouldn't really be there since it does no more than the saver.

v) buffer send <handle>, [<pos>, [<size>]]

This command was used in the above example. It sends the contents of buffer to the destination task. By default the entire buffer is sent but you can specify the start position (pos) and the number of bytes to send (size) if you like.

This command can only be used inside a 'sender' handler when a ram transfer is in progress.

vi) @ext (send)

This variant on @ext is only of use in the 'sender' handler when a ram transfer is in progress. It returns the size of the receive buffer available in the destination application. You may wish to ensure that each 'buffers end' command sends about that many bytes to the destination a,since this is the number it can handle in one go. In general this is not worth worrying about.

vii) archive csv [def] <buffer | file>, [file] <struct> [, <bitstring>] [;]
archive csv [def] <buffer | file>, {value }+ [;]

There are two variants on this command which writes CSV values to a file orbuffer.

<buffer | file> The buffer or file to write to

<struct> The record to write, eg 'video' or 'title'

<bitstring> Defines the fields of the file which should be written. For each field in the file (numbered from 0) there is a bit in the bitstring which determines whether that field should be written.

<value> A value to be written to the file as csv.

The first variant allows writing values from a file's record buffer to a csv file. If <bitstring> is given, it specifies the fields of that file which should be written. Otherwise all fields are written, except the 'system 'field and any external or structure fields.

To use this variant of the command, first set up a bitstring with the fields to be written (eg set bit 5 if you want field 5 to be included) then issue the command.

If you specify [file], every record of the file is written to the csv file.This saves loading in each record one by one and archiving it.

If you specify [def], the field labels are written instead of the values in the record buffer. This is useful since the first line of a csv file should normally give the name of each field.

With a semicolon (;) at the end of the command, a new line will not be started. More values can be then written onto the same line with another 'archive csv' command.

The second variant of the command is much simpler. It just writes the values given in csv format.

For example:

To write a csv file containing all fields of all records of the video 'title' file:

archive csv def fd, title {the field labels }

archive csv fd, file title {the values }

To write fields 4 and 5 of record 10 of the title file:

```
| create a bitstring with bits 4 & 5 set
bit new b
bit set b, 4
bit set b, 5

| archive the two field labels
archive csv def fd, title, b

| load record 10
load title, 10

| & archive the two field values
archive csv fd, title, b
```

To write a string and a number without starting a new line:

```
archive csv fd, "This is a csv string", 12;
```

viii) import csv [def] <buffer | file>, [file] <struct> [, <bitstring>] [;]
import csv <buffer | file>, {| file field}+ [;]

This command is the exact opposite of archive csv. Values can be read from a csv file into variables, file fields or a file's record buffer.

if you specify [def], S-Base will read a line from the file and use this to update the bitstring. For example, imagine a csv file with fields 4 and 5 of the video 'title' file:

```
rating,description
"G",
"PG",
"R14","This movie shows off the latest in special effects. A good story with plenty of action. This is one of the best movies of 1991."
"PG",
"G",
"PG",
```

"R12",
"R16",
"G",
"PG",

If this file is opened and then 'import csv def fd, title, b' executed,where 'b' is a bitstring, S-Base will set bits 4 and 5 of the bitstring b.S-Base reads the first line of the csv file and sees that fields 4 and 5(rating and description) are contained within the file. The rest of the file could then be read using 'import csv fd, title, b' repeatedly until @eof.

Without the semicolon (;) at the end of the command, S-Base will skip to the start of the next line in the file. Otherwise the file pointer will be left as it is.

ix) Buffers and files

Most commands which are described as operating on files in the S-Base Reference can now operate on buffers as well. For example 'print to' will work with a buffer, as will 'archive', 'import', 'set ptr', '@ptr', etc.Buffers have an internal pointer just like files. If data it written to a buffer beyond it's end, the buffer will be extended in 4k chunks.

Reading from and writing to buffers is useful for saving and loading via ram transfer, since the data to send must be present for 'buffer send' and 'buffer import' to work.

x) send to <fname>, <type>, <estsize>, <saver>
[, [<sender>] [, [<printer>] [, <result>]]]

This command is identical to 'save to' except that no dialogue appears. S-Base creates a drag box which the user can drag to save data as with 'save to'. The difference is that the user does not get a chance to enter a filename into a 'save as' dialogue.

This command is useful for implementing drag and drop between fields, or other applications.

xi) @stringize (<array> [, <length> [, <terminator>]]) x)
@stringize (ptr <address> [, <length> [, <terminator>]])

You can now specify a third parameter to @stringize, which is the terminator character to use for the conversion. S-Base always uses ASCII 0 anyway, but you can specify an additional terminator here, such as 13. This makes dealing with (e.g.) template files easier.

The second variant of @stringize allows a memory address to be specified instead of an array. This is useful when the data to be stringized is not in an array (e.g. in a buffer - use @addr (buffer <handle>) to get the buffer address).

xii) import handlers

S-Base v1.0 allows data import by dragging data into your application. To support this you must create a handler in a card or table's handler set of type 'S Import'.

This handler will be called when the user drags a file into the card or table, either from a filer window or from another application. @clickelementlabel can use used to find out which element the user dragged the file onto.

An example import handler is shown below, containing some of the code you would need to place in such a handler:

```
local type, estsize, filename, fd, b, finalsize

{check if a ram transfer is ok. 'load check import' checks if ram transfer is supported by the
sending application, and if so, gets the file type and estimated size of the file to be transferred.
Note that the estimated size may not be accurate }

load check import type, estsize

{turn on the hourglass }
delay on

{create a buffer to hold the imported data }
buffer new b

{ram transfer is not available, 'type' will be -1. This means the sender cannot support ram
transfer and must save to a file. Here we load the data from the file }

if type = -1
  {get the file type and filename }
  load check insert type, filename

  {the file }
```

```
  fd = @openin filename

  {it into our buffer }
  buffer load ext b, fd

  {the file }
  close fd

  {that the file was loaded successfully }
  load insert

  {process the data }
  proc process (b)

{ram transfer is available, so use it }

else

  {the data from the sending application. The estimated size must be given. 'finalsize' is a
variable to receive the final size of the file transferred. If less than zero then an error occurred }
 buffer import b, estsize, finalsize

  {everything went ok with the transfer, process the data }
  if finalsize >= 0
    proc process (b)
  endif
endif

{the buffer containing the loaded data, since it has now been processed }
buffer dispose b

{the hourglass }
delay off
```

The procedure 'process' would have to be written to process the data imported, possibly using import csv. Or perhaps the data is a sprite or drawing which must be saved into an external with 'buffer save'.

You can see that S-Base is handling most of the work. You need only understand a few commands to get data import working, and need not understand the complexities of ram transfer, etc.

If you do not wish to support ram transfer, simply omit the 'load check import' command, and assume that it returned a filetype of -1 (ie start the above code from the 'load check insert' command).

You must always support transfer by file.

xiii) load check import <type>, <estsize>

This command checks if ram transfer is supported by the sending application, and if so, gets the file type and estimated size of the file to be transferred. If not <type> is set to -1 and 'load check insert' should be used.

The estimated size returned is as stated by the sending application, but may not be accurate and may even be zero.

If you do not wish to support ram transfer, then you should not use this function, but simply assume that the sender can't support ram transfer and use 'load check insert' instead.

xiv) load check insert <type>, <filename>

This command indicates to the sending application that your application wishes to load the data via a file. The two parameters are variables which will be set to the file type and filename respectively.

The data should then be loaded from that file and a 'load insert' command issued afterwards.

xv) load insert

This command should be used during data transfer after the file has been loaded. It will cause the sending application to remove the file, since it is finished with.

xvi) buffer import <handle>, <estsize> [, <handler> [, <result>]]

This command was illustrated in the example above. It reads data from a sending application by ram transfer. The amount of data read is determined
by the sending application. <estsize> should be passed as returned from 'load check insert'.

If <handler> is specified, it is a procedure to be called each time new data enters the buffer. This procedure can empty (process) the buffer before the next lot of data is received. A handler need not be specified, and if it is, it need not actually do anything. However, if the handler does empty the buffer, this will reduce the memory requirement of the 'buffer import'.

Without <handler>, all the data will be read into the buffer for processing, which obviously requires more memory. A typical handler might be:

```
def proc handler

{procedure for processing data received in buffer 'b'. It should process whatever data it can
from the buffer, then remove that data, so more data can be appended to the buffer. This
procedure should return TRUE if all is ok, else FALSE }

{call the procedure to process some data. This procedure will attempt to process all the data in
the buffer. It returns the number of bytes processed }

local done = proc process (b, @ext b, count)

{the processed data from the buffer }
buffer delete ext b, 0, done

{ok so return TRUE}
return TRUE

enddef
```

The procedure 'proc process' could do anything, but in this example, it counts the number of 0 bytes in the received data:

```
def proc process (b, size, return count)

local upto
```

The second variant defines a 'return parameter' in a 'def proc' command, as described in the S-Base Reference.

ii) Raw events

Raw event handlers must not 'claim' events (return TRUE) if the event is one which S-Base should see: for example, a mode change message must not be claimed in this way. Neither should a null event be claimed, as this will stop S-Base from operating it's background tasks. In general, claiming raw events should be avoided.

Note that in S-Base 1.0 raw event handlers will be called for any active context (i.e. any running application). This applies even when the application has 'terminated'. Thus you should be careful not to claim raw events which stop S-Base from operating. For example, if you claim all mouse click events it will not be possible to click on list viewers, etc.

If this situation arose because of an error you should be able to remove the context using the icon bar menu, Ctrl-f9 or by loading & running another application. If all else fails, save the application and quit S-Base from the Task Manager Task Display.

iii) pack

As noted in the S-Base Reference, 'pack' does not automatically rebuild the indexes associated with the packed file. You should use 'reindex' for this.

iv) @next and @previous

The S-Base Reference states that these functions return -1 when there is no next or previous record. In fact these functions return 0 in this case, as with @tof and @bof.

v) Scared cards and tables

Where there are two cards (or tables) open and the user clicks in one, the other will receive a 'click' on an element called "<scared>", as noted in the tutorial.

In fact, this 'scared' event can occur when the user presses a key in the other card, or closes the other card. Both actions can compromise scared cards and should cause them to close. S-Base never actually closes a card itself, but relies on the application noticing the scared event.

5. Configuration

Some aspects of S-Base can be configured using the 'Configure->S-Base' option on the icon bar menu, or manually using the file called 'config' within the S-Base directory.

The various configuration options available are listed below. The first seven are available from within S-Base, the others can only be altered by editing the 'config' file.

The configuration options are listed by their label, which precedes the value for that option in the configuration file.

i) drawfile_rubouts

S-Base has an option for converting a page of a report to a drawfile. Normally, text in the resulting drawfile does not overlap. Sometimes it does and in this case it is useful to put a 'rubout' rectangle behind each text object so that it overwrites any overlapping text, rather than just being merged on top of it.

Setting this option on will increase the size and alter the structure of drawfiles generated by reports.

ii) clip_printouts

This option controls the clipping of reports when sent to the printer driver. You should normally leave this option on, as otherwise reports will not be printed as shown on the screen. This option allows you to see how a report will look when exported as a drawfile (since these are not clipped).

iii) compacted_archives

Application archives (wihin the 'archive' text file) are always larger than the binary application definition, even though the two are equivalent. S-Base employs a compaction method which converts the archives to a less readable but more compact form. This takes slightly longer to generate and uses about half the space of an uncompacted archive.

If you wish to read an archive, or perhaps alter it using a text editor, you may wish to use an uncompacted archive since it is easier to follow.

```
| work through each byte of the buffer checking for 0 ffor upto = 0 to size - 1
  if @ipeek (@addr (buffer b) + upto, 1) = 0 then count += 1
next

{the number of bytes processed (= all of them) }
return size

enddef
```

The code to drive the above handler might be:

```
local type, estsize, b, finalsize, count = 0

| Import csv file into browser

load check import type, estsize

buffer new b

if type = -1
  ...
else
  buffer import b, estsize, finalsize, handler
  if finalsize > 0 then proc process (b, finalsize, count)
endif

print count, " zero bytes found"

buffer dispose b
```

This example prints to the command line the number of zero bytes found in the input. Note that the final 'proc process' is necessary to process the final instalment of data, which might not have filled the buffer.

If finalsize returns < 0, then the transfer failed.

xvii) report run [print] <handle> [, <flags> [, <copies>]]

This command allows reports to be sent to the printer without the user printing each page individually. The following values are defined for <flags>:

0 : Ask the user how many copies to print.
1 : Don't ask, just use <copies>, or 1 if not specified.

Set <flags> to 1 if you don't want to prompt the user to print the report. There should normally be some way for the user to select printing to 'screen' or 'printer'. To print to the printer only, without first displaying the report on the screen, simply omit the 'report show' command before 'report run'.

<copies> defaults to 1, but an alternative value can be specified.

3. Deletions

i) The 'windup' program.

The S-Base Tutorial states that a program called 'windup' will be executed when the application exits. This feature is not implemented in S-Base 1.0. You are unlikely to find a need for this feature.

4. Comments

i) return [<expr>]
return <var>

This command was omitted from the S-Base Reference. It's use is documented in full in the Tutorial however.

The first variant returns from a procedure or handler, with an optional expression as the 'return value'. This can appear anywhere in the procedure or handler and immediately called it to be exited. If the procedure is called in an expression (so that a return value is required), or the handler expects a return value, then this command must be used to exit the procedure, or the error 'Return value expected' will be generated.

Note that S-Base 1.0 does not use text compression on archives. Compacted archives are still 7-bit ASCII.

iv) minimise_memory

Generally when S-Base loads in an object it will stay loaded. With this option set, S-Base will throw away objects where possible. This reduces the speed of operation (sometimes considerably) but reduces the memory requirements of S-Base.

It is recommended that this option be set 'off' in normal use.

v) kerning

S-Base supports kerning which can be turned on or off using this option. When on, all text displayed by S-Base, in cards, tables and reports will be kerned, provided kerning information is available with the font used.

This features is available when using the RISC OS 3 ROM fonts, for example.

vi) fast_desubscript

This option speeds up array accesses by caching information relevant to the access. This requires a (very) small memory overhead so you might want to turn it off if memory is tight.

vii) raw_events

Normally, non-window raw events are passed to an handler set called 'raw', if it exists in the application. There is a considerable overhead involved in passing all events received by S-Base to this handler set, and most application will not need this feature.

For this reason you may wish turn off non-window raw events by setting this option off.

viii) autoscroll_delay

This is the delay in centiseconds (1/100 s) before a window will autoscroll. You can increase this if you feel S-Base autoscrolls windows too readily.

ix) autoscroll_repeat

This is the delay between each window autoscroll (assuming your machine is fast enough to keep up). You can increase this if you feel that S-Base autoscrolls windows too quickly.

Control over autoscrolling is provided so that it does not become unusably fast on fast machines. You can disable autoscrolling altogether by holding down the <Ctrl> key while dragging objects around.

x) toolbar_type

This should always be set to 1. It controls the toolbar in the template editor.

xi) network

This option only applies to S-Base Developer Plus.

xii) server

This option only applies to S-Base Developer Plus.

xiii) net_servicelist_time

This option only applies to S-Base Developer Plus.

xiv) layout_info_xoffset

This is the default offset in OS units from the left of a format or template layout window to the left of the info bar. You can alter this if you feel the info bar is inconveniently positioned.

xv) layout_info_yoffset

This is the default offset in OS units from the bottom of a format or template layout window to the bottom of the info bar. You can alter this if you feel the info bar is inconveniently positioned.

xvi) trace

This option is for tracing using a spy application, and is of no use to users of S-Base.

xvii) errfile

This option turns on logging of errors within S-Base, and is of no use to users of S-Base.

xviii) layout_detail_delay

This is the delay in centiseconds (1/100 s) before the 'operation detail' part of the info bar is cleared. For example, when set to 200 and an object is deleted, S-Base will wait for two seconds before clearing the message 'Object ... deleted' from the info bar.

xix) menu_label_length

S-Base truncates long labels in the viewer windows. You can increase this value if you feel that S-Base is being too severe with the truncation.

xx) max_users

This option only applies to S-Base Developer Plus.